



Persistent

Core Java: Object Oriented Programming-I

Persistent Interactive | Persistent University



Key learning points:

- At the end of this module, you will be able to understand & implement :
 - Encapsulation
 - Abstraction
 - Inheritance
 - Polymorphism
 - Creating classes and instances
 - Constructors and Initialization
 - Reference Variable casting
 - Static data and methods
 - Abstract classes and Interfaces
 - Enums

Object oriented program (OOPS)

- What is OOP?

An Object-Oriented Program consists of a group of cooperating objects, exchanging messages, for the purpose of achieving a common objective.

- Benefits of OOPs:

- Real-world programming
- Reusability of code
- Modularity of code
- Resilience to change
- Information hiding

Object Oriented Programming Concepts

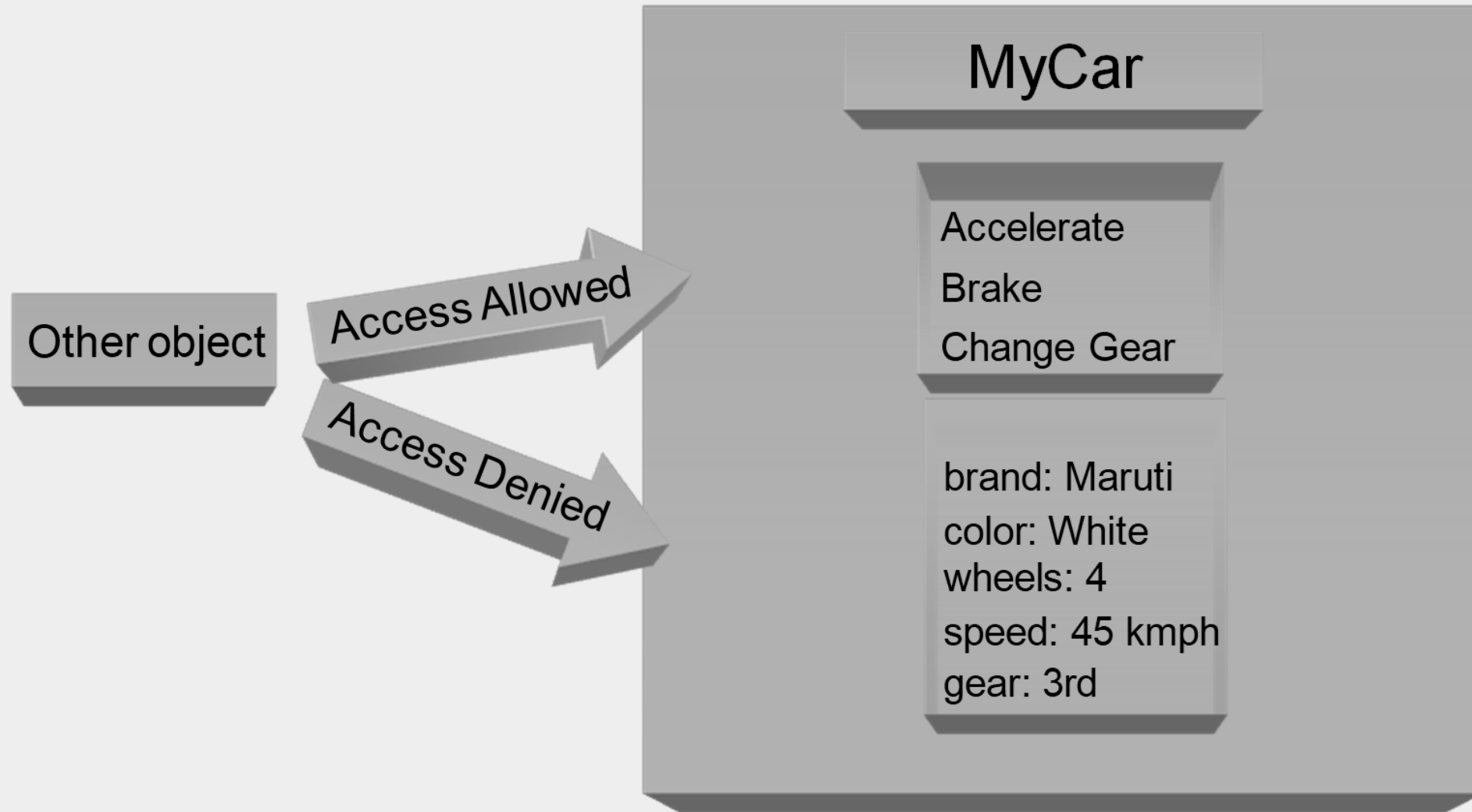
- Below are the Object Oriented Programming Concepts
 - Encapsulation
 - Abstraction
 - Inheritance
 - Polymorphism

Encapsulation

- Encapsulation is the ability of an object to place a boundary around its properties (i.e. data) and methods (i.e. operations).
- Grady Booch, defined the encapsulation feature as:

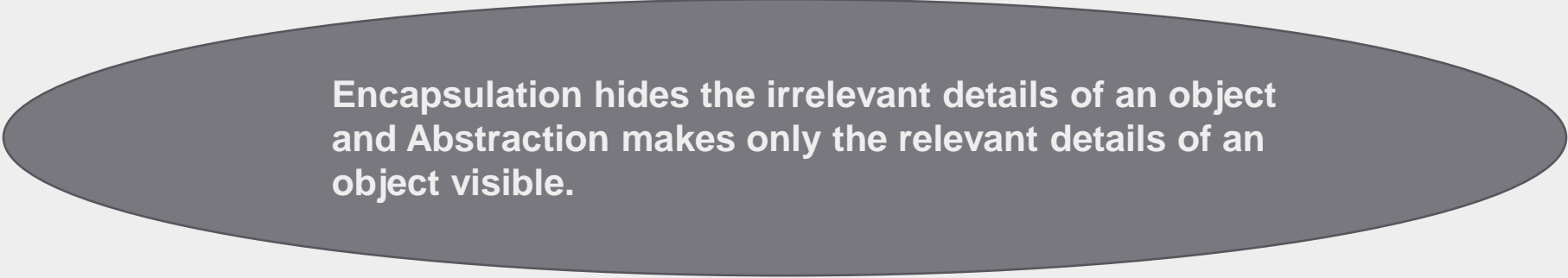
“Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics.”

Example: Encapsulation



Abstraction

“An Abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer.”

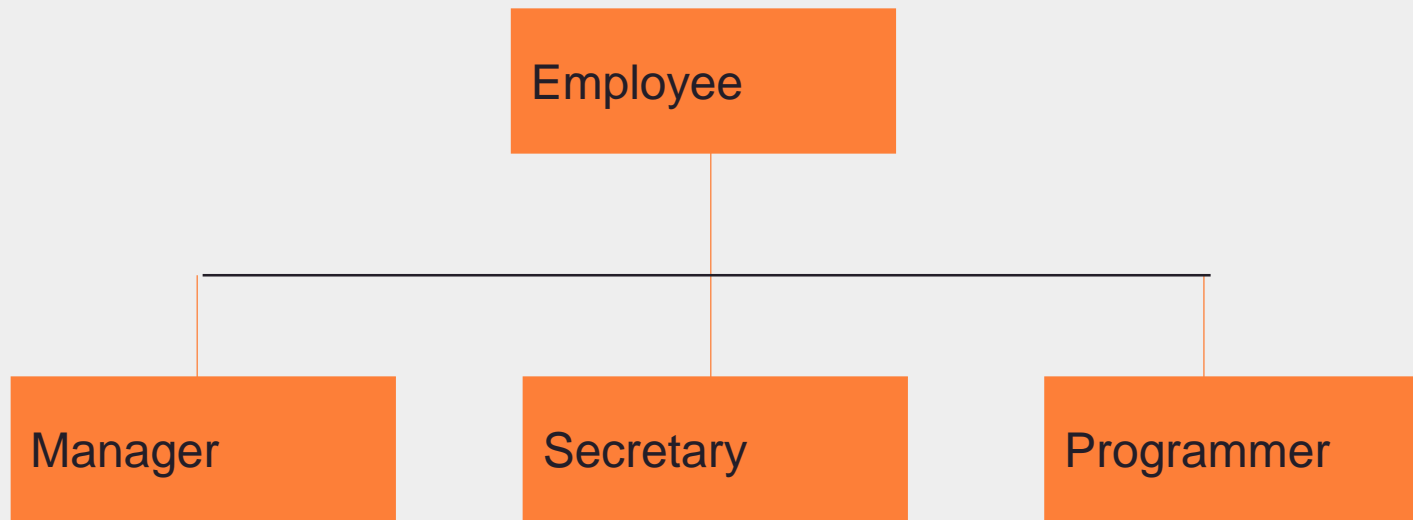


Encapsulation hides the irrelevant details of an object and Abstraction makes only the relevant details of an object visible.

Inheritance

- Inheritance is the capability of a class to use the properties and methods of another class while adding its own functionality.
- Enables you to add new features and functionality to an existing class without modifying the existing class.
- Superclass and Subclass
 - A superclass or parent class is the one from which another class inherits attributes and behavior.
 - A subclass or child class is a class that inherits attributes and behavior from a superclass.

Inheritance (continued)

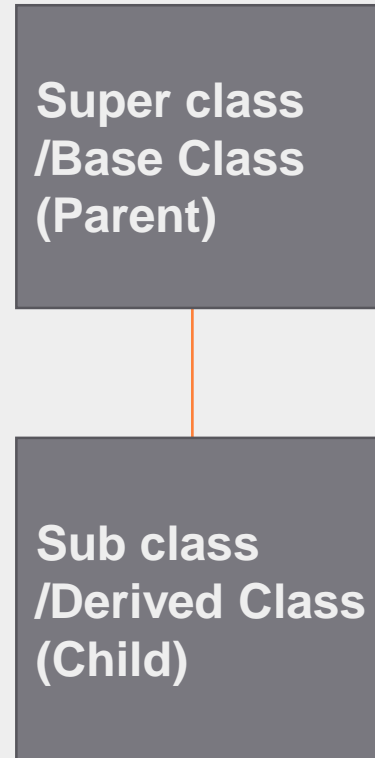


Types of inheritance

- Two types of inheritance are
 - Single inheritance
 - Multiple inheritance

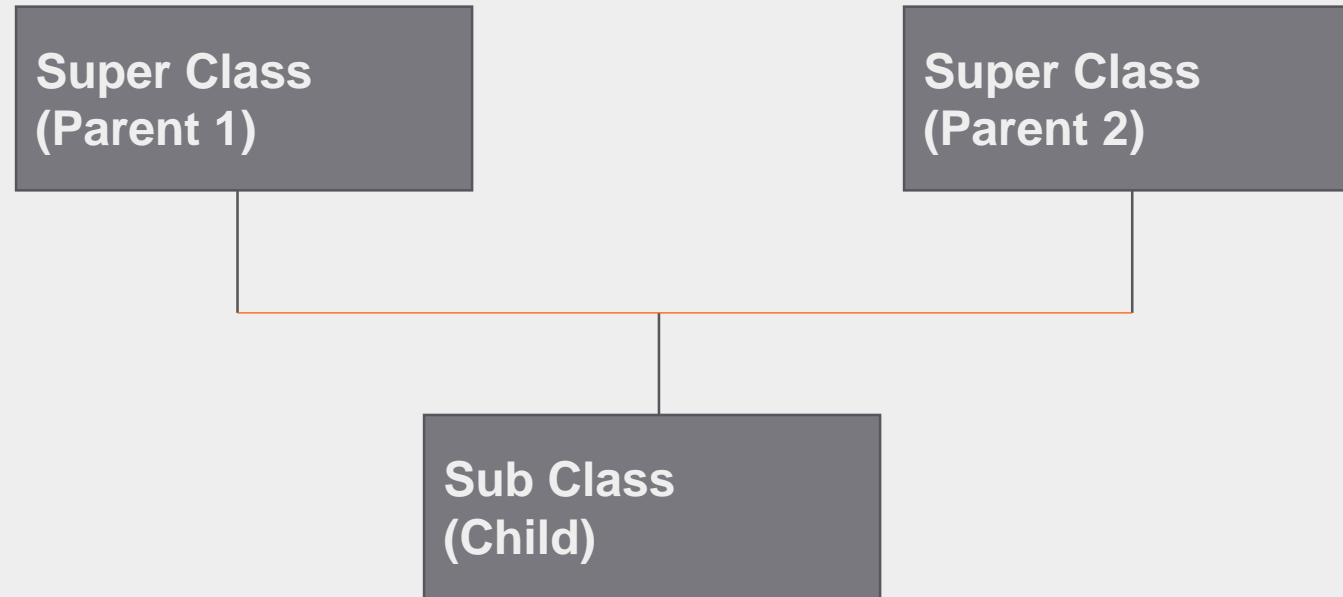
Single inheritance

- Subclass is derived from only one superclass.



Multiple inheritance

- A subclass is derived from more than one super class. (Java does not support multiple inheritance)



Polymorphism

- Derived from two Latin words - Poly, which means many, and morph, which means forms.
- It is the capability of an action or method to do different things based on the object that it is acting upon.
- In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class.

Types of Polymorphism

- Two types of polymorphism are
 - Compile time polymorphism
 - Runtime polymorphism

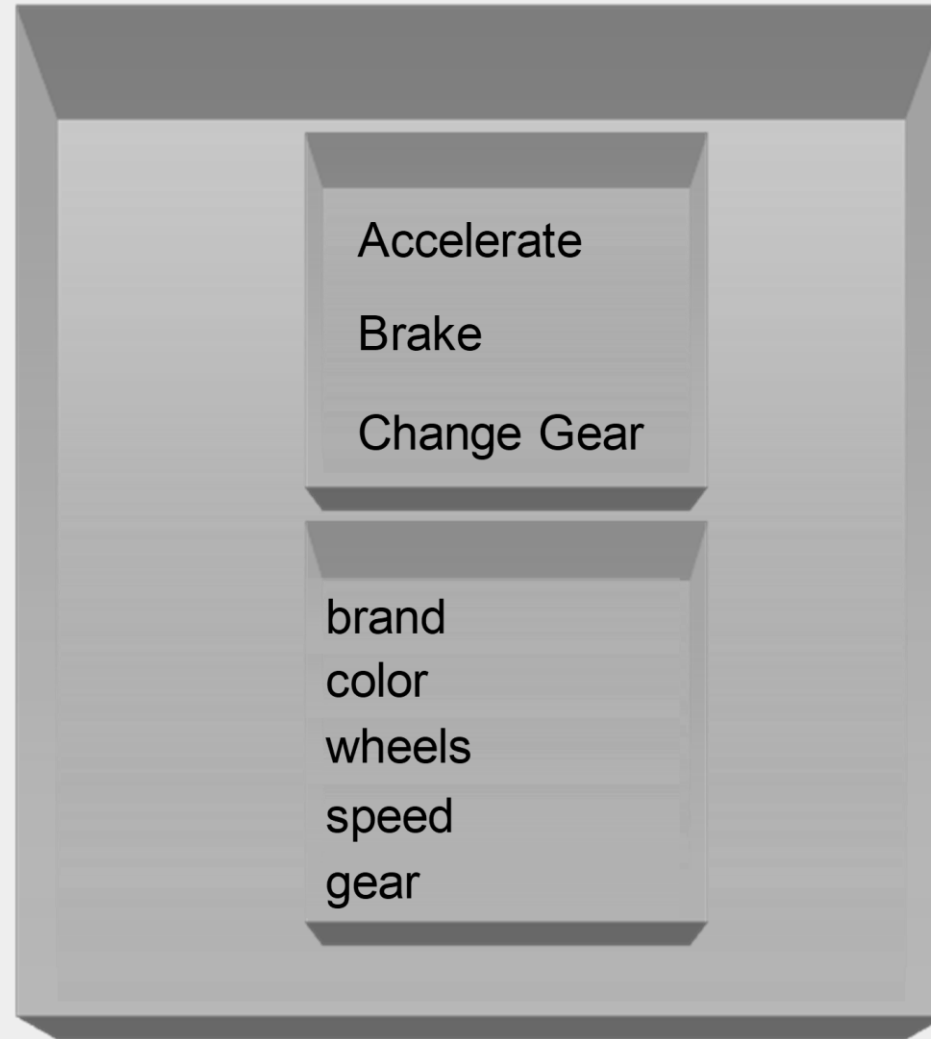
Structure of Java application

- A class consists of
 - Fields i.e. data members
 - Methods i.e. functions

What is a class?

- A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.
 - blueprint: A class can't do anything on its own.
 - defines: A class provides something that can be used later.
 - objects: A class can only be used, if it had been "brought to life" by instantiating it.

Example: Class



Object

- An object is a software construct that encapsulates data, along with the ability to use or modify that data, into a software entity.
- An object is a self-contained entity which has its own private collection of properties (i.e. data) and methods (i.e. operations) that encapsulate functionality into a reusable and dynamically loaded structure.

Object (continued)

- Booch defines an object as: “Something you can do things to”.
- An object has:
 - state,
 - behavior, and
 - identity
- The structure and behavior of similar objects are defined in their common class.”

Example: Object

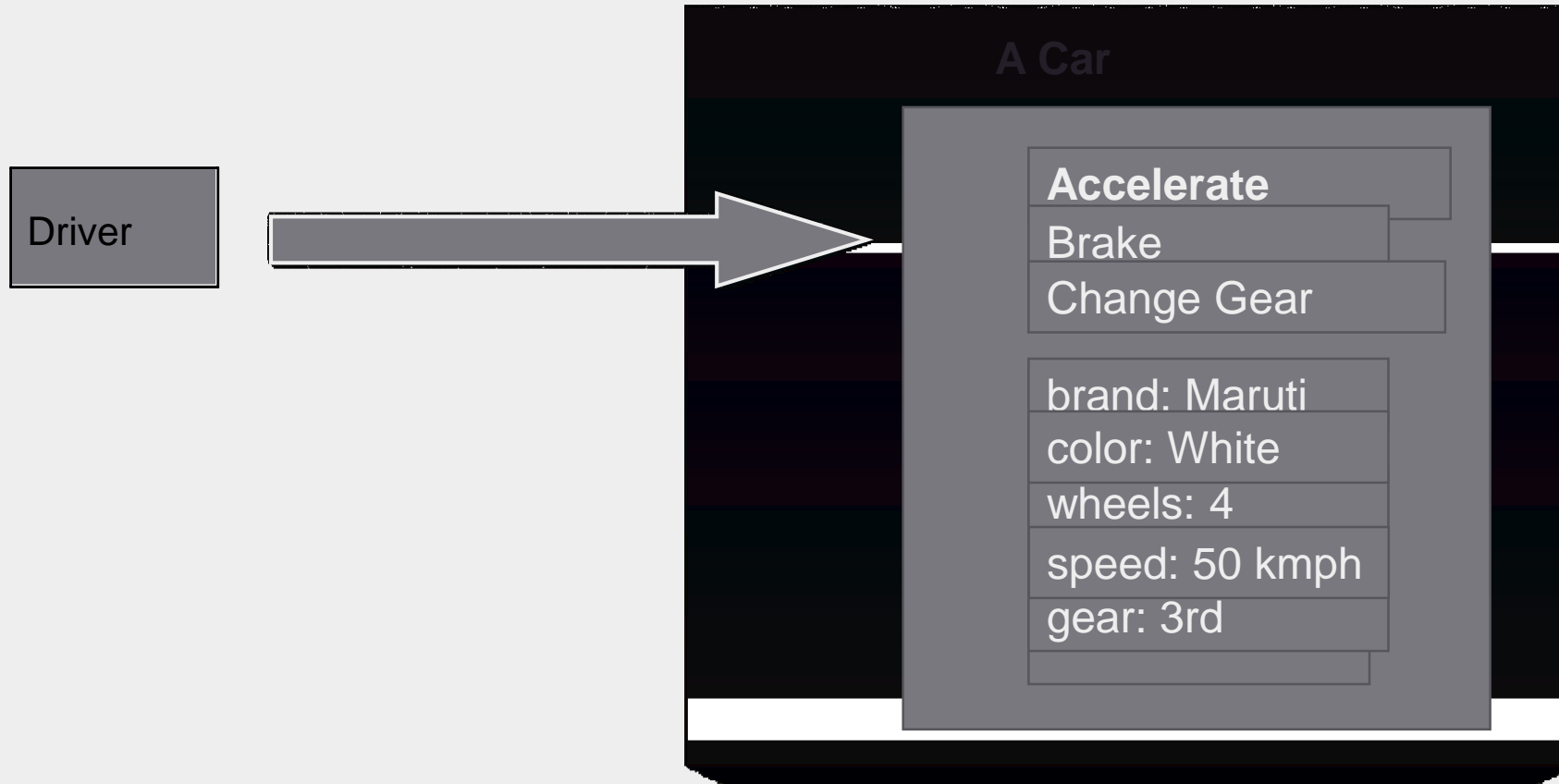


Methods

- An operation upon an object, defined as part of the declaration of a class.
- The methods, defined in a class, indicate, what the instantiated objects are able to do.

Example: Methods

- Driver wants to increase the speed of the car?



Java class definition: Syntax

- This is the syntax to write a class

```
[modifier] class classname
{
    // Variables definitions
        modifier datatype data_variable1;
        modifier datatype data_variable2;
        .....
    // Method definitions
        modifier method1(datatype parameter11, datatype
        parameter12,..)
        {
            //method definition
        }
        modifier method2(datatype parameter21, datatype
        parameter22,....)
        {
            //method definition
        }
        .....
}
```

Structure of Java application - Example

```
class Car
```

```
{
```

```
    String brand;
```

```
    String color;
```

```
    int wheels;
```

```
    int speed;
```

```
    int gear;
```

```
    void accelerate(){}  
    void brake(){}  
    void changeGear(){}  
}
```


Return Type

Method Name

Structure of Java application (continued)

- This is syntax to create an object

```
Car myCar1 = new Car();
```



**Class name
or the
default
constructor**

Constructors

- A constructor is a special method that initializes the instance variables. The method is called automatically when an object of that class is created.
- A constructor method has the same name as the that of the class.
- A constructor always returns objects of the class type hence there is no return type specified in its definition.

Constructors

- A constructor is most often defined with the accessibility modifier “public” so that every program can create an instance but is not mandatory.
- A constructor is provided to initialize the instance variables in the class when it is called.

Constructor

- If no constructor is provided, a default constructor is used to create instances of the class. A default Constructor initializes the instance variables with default values of the data types.
- If at least one constructor is provided , default constructor is not provided by JVM.

Overloaded Constructors

- A class can have more than one constructors, provided that the parameters passed to each constructor are different.

```
public class Box {  
    int x;  
    int y;  
    Box() {  
        int x = 0;  
        int y = 0;  
    }  
    Box(int x; int y ){  
        this.x = x;  
        this.y = y;  
    }  
}
```

//if class has explicit constructor so cannot rely on default one.

```
Box b1 = new Box(10,10);  
Box b2 = new Box();
```

The *this* reference

- “this” is a reference to the current object.
- This is used to
 - refer to the current object when it has to be passed as a parameter to a method.
 - Refer to the current object when it has to be returned in a method.
 - Refer the instance variables if parameter names are same as instance variables to avoid ambiguity.

```
    this.name=name;  
}
```

```
Otherobj1.anymethod(this);
```

```
method1(String name){  
    this.name = name;  
}
```

The *this* reference (continued)

- The `this` keyword included with parenthesis i.e. `this()` with or without parameters is used to call another constructor. The default construct for the `Car` class can be redefined as below
- The `this()` can be called only from a constructor and must be the first statement in the constructor

```
class Car
{
    String brand,color;
    public Car(){

        this("NoBrand","NoColor",4,0);
    }
    public Car(String brand,String color)
    {

    }
}
```

Summary :

With this we have come to an end of our session, where we discussed about

- Object Oriented Programming concepts
- Basic constructs of Java as classes, objects, constructors, methods

At the end of this session, we see that you are now able to

- Create classes
- Create objects
- Access data members & methods using class object

Appendix

A decorative graphic consisting of a horizontal orange line that extends from the left edge of the slide. This line meets a vertical orange line that extends downwards to the bottom edge. At the intersection, a large orange circle is drawn, with its center at the intersection point. The circle's top edge is near the top of the slide, and its right edge is near the right edge of the slide.

References

Thank you

Reference Material : Websites & Blogs

- <https://docs.oracle.com/javase/tutorial/java/concepts/>
- <http://www.javatpoint.com/java-oops-concepts>
- http://www.tutorialspoint.com/java/java_overview.htm

Reference Material : Books

- **Head First Java**
 - By: Kathy Sierra, Bert Bates
 - Publisher: O'Reilly Media, Inc.
- **Java Complete Reference**
 - By Herbert Schildt



Thank you!

Persistent Interactive | Persistent University

